

# Wearable Device Bluetooth/BLE Physical Layer Dataset

Artis Rusins <sup>1,†</sup> , Deniss Tiscenko <sup>1,†</sup> , Eriks Dobelis <sup>2,†</sup> , Eduards Blumbergs <sup>1,†</sup> , Krisjanis Nesenbergs <sup>1,\*,†</sup>   
and Peteris Paikens <sup>2,†</sup> 

<sup>1</sup> Institute of Electronics and Computer Science, 14 Dzerbenes St., LV-1006 Riga, Latvia; artis.rusins@edi.lv (A.R.); e.blumbergs@gmail.com (E.B.)

<sup>2</sup> Institute of Mathematics and Computer Science, University of Latvia, Raina blvd. 29, LV-1006 Riga, Latvia; peteris@ailab.lv (P.P.)

\* Correspondence: krisjanis.nesenbergs@edi.lv

† These authors contributed equally to this work.

**Abstract:** Wearable devices, such as headsets and activity trackers, rely heavily on the Bluetooth and/or the Bluetooth Low Energy wireless communication standard to exchange data with smartphones or other peripherals. Since these devices collect personal health and activity data, ensuring the privacy and security of the transmitted data is crucial. Therefore, we present a dataset that captures complete Bluetooth communications—including advertising, connection, data exchange, and disconnection—in an RF isolated environment using software-defined radio. We were able to successfully decode the captured Bluetooth packets using existing tools. This dataset provides researchers with the ability to fully analyze Bluetooth traffic and gain insight into communication patterns and potential security vulnerabilities.

**Dataset:** <https://pubfaii.edi.lv/wearsecdata>

**Dataset License:** CC-BY-SA

**Keywords:** RF; PHY layer; SDR; wireless; Bluetooth; BLE; wearable devices



**Citation:** Rusins, A.; Tiscenko, D.; Dobelis, E.; Blumbergs, E.; Nesenbergs, K.; Paikens, P. Wearable Device Bluetooth/BLE Physical Layer Dataset. *Data* **2024**, *9*, 53. <https://doi.org/10.3390/data9040053>

Academic Editor: Giuseppe Ciaburro

Received: 27 February 2024

Revised: 22 March 2024

Accepted: 29 March 2024

Published: 3 April 2024



**Copyright:** © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Summary

Bluetooth is a very popular communication standard that is used to exchange data between devices over short range. It is mostly used by various types of wearable devices, smartphones, and computer peripherals. There are many existing studies that outline both the privacy and security risks with this standard, most notably the possibility to fingerprint the devices' radio frequency (RF) waveform, thus breaking Bluetooth's integrated privacy mechanisms [1–3], and the use of many successful fuzzing tools [4], which has serious security implications. To perform this kind of cybersecurity research one needs access to Bluetooth traffic. Specifically, RF fingerprinting requires working with raw RF waveforms, so we cannot use off-the-shelf Bluetooth dongles because those will output already processed (demodulated, decoded, etc.) RF data. Because both Bluetooth Classic (BTC) and Bluetooth Low Energy (BLE) operate in the 2.4 GHz band, which is also used by several other wireless technologies, such as Wi-Fi and Zigbee, it is difficult to isolate RF signals from the one Bluetooth connection we are interested in. One approach would be to try to decode all traffic and distinguish packets sent by their broadcast address (BD\_ADDR). One widely used low-cost approach involves using the Ubertooth One board by Great Scott Gadgets, but since it uses an embedded RF (radio frequency) transceiver with a maximum bandwidth of 1 MHz [5,6], it can effectively only receive one Bluetooth channel at a time and cannot sniff all the packets presented. Another problem with the “decode everything we receive” approach is that even if we decode the BD\_ADDR, we cannot be sure that it originates from the Bluetooth device we are interested in testing

and not from something else nearby, since according to the Bluetooth SIG standard, the BD\_ADDR must be randomized. Another prominent tool is the ice9-bluetooth-sniffer [7], which uses software-defined radio (SDR) to record raw radio signals and decode the entire received bandwidth, thus not missing any packets, but it still does not solve the problem of distinguishing between multiple devices which are using randomized BD\_ADDR. In order to analyze the RF waveforms of a particular Bluetooth connection, it is necessary to isolate the devices of interest from the rest of the RF spectrum.

Several publicly available datasets capture Bluetooth packets in isolated environments, such as described by E. Uzundurukan, Y. Dalveren, A. Kara in “A Database for the Radio Frequency Fingerprinting of Bluetooth Devices” [8], where the researchers collected data from 27 different smartphones using a high sample rate oscilloscope. The dataset is available for download as .txt files containing voltage readings. Another dataset by A. Siddik et al., “Wideft: A Corpus Of Radio Frequency Signals For Wireless Device Fingerprint Research” [9], includes not only Bluetooth data but also encompasses WiFi and other devices. In the “Wideft” dataset, the data are pre-processed by extracting energy bursts and stored in 16-bit samples, reducing the overall dataset size by eliminating inter-packet noise by skipping noise between packets. Both of these datasets solely encompass advertising data from Bluetooth devices and do not aim to capture a whole data exchange. A. Jagannath et al. have published two physical layer Bluetooth datasets as part of RF fingerprinting research [2,3], which capture “real-world” wireless traffic, as published in IEEE DataPort [10,11]. However, these datasets were collected in non-isolated indoor laboratory settings with emissions from other nearby devices and lacking controlled inter-device communication. To our knowledge, our dataset is the only publicly available dataset that captures the entire physical layer of Bluetooth communication between two devices in an isolated environment with controlled communication, without any destructive processing of the raw recording data.

The aim of this work is to publish a physical layer recording dataset of different wearable devices connecting to an Android smartphone in an isolated environment, thus providing researchers with the ability to fully analyze Bluetooth traffic without any external RF interference. Each recording is performed in an anechoic chamber and a full description of the recording is provided in the metadata.

The dataset presented consists of 32 of the most popular Bluetooth wearable device RF recordings, described in Section 3.2, and also an additional recording without any devices to capture the baseline noise level in an anechoic chamber, and one recording with Bluetooth data from a smartphone used as the master device in the remaining recordings—the Samsung Galaxy S20 FE. Each recording ranges from approximately 10 to 30 s in duration, with a sample rate of  $100 \times 10^6$  and a data type of `numpy.complex64`. Timestamps for each part of the communication are included in the metadata file. For our dataset, a sample of the most popular wearable devices and device types was selected. Each device has two recordings (experimental trials) and each recording consists of two scenarios: paired and unpaired. Each radio recording consists of four phases as follows:

1. Advertising, during which only the device under test (DUT) transmits;
2. Pairing, when the DUT initiates pairing with an Android smartphone;
3. Data exchange, involving tasks such as audio playback or sensor reading;
4. Disconnect, when the Bluetooth of the Android smartphone is deactivated.

In our metadata, “paired” and “unpaired” denote whether the DUT was previously paired with an Android smartphone.

The data acquisition process utilized an Ettus Research USRP X310 [12] equipped with a CBX-120 daughterboard [13]. The sample rate was set to 100 MHz, with each sample composed of in-phase (I) and quadrature (Q) samples combined in a complex float variable format (`numpy.complex64`). The collected data are subsequently channelized into smaller bandwidth files for further processing. Following channelization, we conduct waveform detection, demodulation, and decoding in accordance with the Bluetooth standard. The

decoding results are stored in separate files, occupying significantly less disk space. This allows analysis of Bluetooth traffic without having to deal with the physical layer data.

Because the recorded dataset is too large for common file sharing options such as Zenodo, which offers up to 50 GB of storage per dataset [14], we opted to host the dataset ourselves and make it public using a Nextcloud client, accessible through <https://pubfaili.edi.lv> (accessed on 22 February 2024).

The rest of this document is structured as follows: Section 2 describes the dataset as such, Section 3 discusses the methods used to acquire the data as well as data validation and quality, and finally, Section 4 contains some practical notes on using the data.

## 2. Data Description

The folder structure of the dataset is shown in Figure 1 and is the same for each wearable DUT:

```

'-- Device_name
  |-- recording_1
  |   |-- paired
  |       |-- process
  |           |-- radio_25_n.chdata
  |           |-- radio_05_n_m.chdata
  |           |-- radio.data
  |           '-- top.yaml
  |-- unpaired
  |       |-- process
  |           |-- radio_25_n.chdata
  |           |-- radio_05_n_m.chdata
  |           |-- radio.data
  |           '-- top.yaml
'-- recording_2
  |-- paired
  |   |-- process
  |       |-- radio_25_n.chdata
  |       |-- radio_05_n_m.chdata
  |       |-- radio.data
  |       '-- top.yaml
  |-- unpaired
  |       |-- process
  |           |-- radio_25_n.chdata
  |           |-- radio_05_n_m.chdata
  |           |-- radio.data
  |           '-- top.yaml
'-- _decoded_data
  |-- device_name_recording_k_scenario.json

```

**Figure 1.** Example of a dataset folder structure for a single DUT.

As mentioned above, each device has two recordings (experimental trials) each consisting of *paired* and *unpaired* scenarios. In the context of our dataset, *unpaired* denotes that the device under test (DUT) was not previously paired with the Android smartphone, whereas *paired* signifies that the DUT had been previously paired and can be observed under “Paired” in the smartphone settings. The inclusion of both scenarios is crucial as it triggers different security key exchange mechanisms within the Bluetooth protocol stack [15], which could provide valuable insights for cybersecurity research. We did not implement a paired/unpaired split when collecting data from smartwatches. Instead of using regular Bluetooth connections to the smartphone, smartwatches utilize their specific applications to discover and connect to nearby devices.

For further processing, the *radio.data* is channelized into multiple channels, and the results are stored in the *process* folder. Each *process* folder comprises the outputs of two stages of channelization. In the first stage, the *radio.data* is channelized into four 25 MHz channels. In the second stage, each 25 MHz channel is further divided into five 5 MHz channels, resulting in a total of twenty 5 MHz channels. This processing is described in detail in Section 4.

The data themselves are described by the following files:

- *radio.data* - IQ data from SDR, with a bandwidth of 100 MHz
- *top.yaml* - YAML-formatted file containing metadata
- *radio\_25\_n.chdata* - Channelized versions of *radio.data* into four 25 MHz channels, where  $n = \{0, 1, 2, 3\}$
- *radio\_05\_n\_m.chdata* - Each channel from *radio\_25\_n.chdata* is further divided into five 5MHz channels,  $m = \{0, 1, 2, 3, 4\}$
- *device\_name\_recording\_k\_scenario.json* - Demodulated and decoded Bluetooth data for each radio recording, saved in JSON format. The filename includes the device name, recording index  $k$  (1 or 2), and the scenario (paired or unpaired).

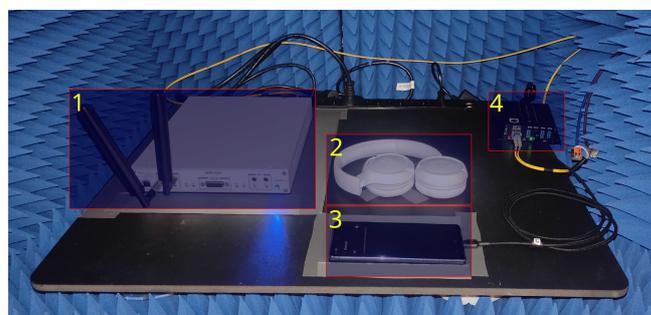
### 3. Methods

To acquire this dataset, we initially defined the scope of the problem as follows: To our knowledge, there is no publicly available dataset of complete BLC and/or BLE communication between two devices recorded in an isolated environment. We captured all phases of Bluetooth communication: advertising, connection establishment, data exchange, and disconnection. This was done because each of these communication phases utilizes different types of data packets, including ID, POLL, FHS, etc. [16]. We wanted the dataset to have a diverse range of packet types for a comprehensive analysis.

To achieve this goal, we devised a data acquisition setup as described in Section 3.1, then selected the devices for data acquisition as described in Section 3.2, and used the data acquisition methodology for each of these devices as described in Section 3.3, resulting in the final published dataset, the data quality of which we validated as described in Section 3.4.

#### 3.1. Data Acquisition Setup

All of the recordings were performed in an anechoic chamber. The Android smartphone was connected to the PC via an optically decoupled USB hub, while the USRP X310 SDR was connected to the same PC via an SFP+ optical transceiver and an optical cable to minimize external interference. For each experiment, everything remained static and only the DUT changed. The setup is shown in Figure 2. The PC outside the chamber controls the Samsung Galaxy S20 FE smartphone via a USB connection and stores data from SDR to disk using Gnuradio software (version 3.10) [17].



**Figure 2.** Anechoic chamber with recording setup. Parts of the setup are marked in blue. The USRP X310 (1) serves as a recording device of the Bluetooth connection between the DUT (2) and the Samsung Galaxy S20 FE (3). (3) is connected to the PC via an optically decoupled USB hub (4) while (1) is connected to the same PC via an SFP+ optical transceiver and an optical cable.

### 3.2. Device Selection

As the recording device, we used the SDR USRP X310 with the CBX-120 daughterboard due to its capability to capture all Bluetooth channels simultaneously. For the Bluetooth master device, we selected the Samsung Galaxy S20 FE, given its status as a relatively modern smartphone at the time of the dataset recording. We used our limited funds to maximize the user base covered by our device list. To compile our list of wearable Bluetooth devices, we initially identified the most popular device categories by examining current Google Trends over the past five years. According to our analysis, headphones emerged as the most popular category, followed closely by activity trackers (inclusive of smartwatches). We then selected the top-selling devices in these two categories on Amazon at the time. The resulting list of devices is detailed in Table 1. Some of the system-on-chip (SoC) chipsets listed there were identified through physical disassembly of the devices followed by direct examination of the chipset inscriptions under a microscope.

**Table 1.** Devices in the dataset.

Folder Name	Class	Bluetooth Version	Chipset
Amazfit_Band_5	Activity Tracker	5.0	not disclosed
Apple_AirPods_(3rd_generation)	Headset	5.0	Apple H1
Apple_AirPods_Pro_(2nd_generation)	Headset	5.3	Apple H2
Apple_Watch_SE_(2nd_Gen)	Activity Tracker	5.3	Apple S5
Apple_Watch_Series_8	Activity Tracker	5.3	Apple S8
Beats_Solo3_Wireless	Headset	4.0	Apple W1
Bose_QuietComfort_Earbuds_II	Headset	5.3	Qualcomm QCC5171
eSense	Headset	-	not disclosed
Fitbit_Charge_5	Activity Tracker	5.1	not disclosed
Fitbit_Versa_4	Activity Tracker	5.2	not disclosed
Garmin_Instinct_Crossover	Activity Tracker	5.0	not disclosed
Garmin_Venu_SQ	Activity Tracker	5.0	Nordic Semiconductor nRF52810
Garmin_Vivoactive_4	Activity Tracker	5.0	not disclosed
Google_Pixel_Buds_Pro	Headset	5.3	Broadcom BCM43015A0WKUBG
Google_Pixel_Watch	Activity Tracker	5.2	Exynos 9110+Cortex M33
Huawei_Band_3e	Activity Tracker	4.2	Ambiq Micro Apollo3 Blue
I7-TWS	Headset	-	not disclosed
JBL_TUNE510BT	Headset	5.0	Realtek RTL8763B
Unknown_BT_headphones_black	Headset	-	not disclosed
Mangoman	Headset	-	not disclosed
noise	-	-	-
Raycon_The_Everyday_Earbuds	Headset	5.0	Airoha AB1562M
Redmi_Buds_3	Headset	5.0	not disclosed
Samsung_Galaxy_Buds2_Pro	Headset	5.3	BES BES2700YP
Samsung_Galaxy_S20_FE	Smartphone	5.0	not disclosed
Samsung_Galaxy_Watch5	Activity Tracker	5.2	Exynos W920
Smart_Bracelet_LP715(G)	Activity Tracker	4.0	not disclosed
Smart_Bracelet_XMSH07HM	Activity Tracker	4.0	not disclosed
Sony_WF-1000XM4	Headset	5.2	MediaTek MT2822SA
Sony_WH-1000XM5	Headset	5.2	MediaTek MT2822AA
Xiaomi_Smart_Band_7	Activity Tracker	5.2	Dialog DA14706
ZABBOW_Scorpion	Headset	-	not disclosed

### 3.3. Acquisition Methodology

Each radio recording follows the following methodology:

1. Turn on DUT in advertising mode and position it within the anechoic chamber.
2. Start SDR in receive mode.
3. Enable Bluetooth on the Android device and establish a connection to the DUT using the Android Debug Bridge (ADB) [18].
4. Exchange data with the DUT; for headphones, initiate audio playback, while for smartwatches/trackers, interact with specific apps to trigger sensor readings.
5. Disable Bluetooth on the Android device.
6. Terminate SDR receiving.
7. Write metadata.

We start each recording with the DUT already in advertising mode and placed inside the chamber, as activating the DUT often necessitates physical access. Through the ADB interface, we manage the remaining interactions, allowing us to timestamp critical steps within the communication process with reasonable accuracy as follows: enabling Bluetooth on the smartphone, establishing a Bluetooth connection, initiating data exchange, terminating data exchange, and disconnecting. This approach captures the entire communication between devices similarly to how it would happen in a real life scenario. Recording metadata keys are described in Table 2.

**Table 2.** Metadata keys and their descriptions.

Key	Description
recording_date	The date the radio recording was made.
recording_location	Location where the radio recording was made.
recording_device	Parameters regarding the recording device.
device_type	Type of device involved, for example, SDR.
model	Model of device_type.
daughterboard	RF Frontend, USRP's use term daughterboard.
attenuator	External attenuator (if) used.
antenna	Antenna used.
uhd_version	Driver version for SDR.
sample_rate	Sample rate used.
center_frequency	Center frequency, Hz.
gain	Internal gain value for the recording device.
DC_correction	Whether DC correction was used or not. In USRP UHD driver, this is "uhd.tune_request()"
output_file	Output recording file with reference to metadata location.
wearable_device	Parameters regarding the wearable device (DUT).
device_type	Device type: Smartwatch/smartband or headphones, or smartphone.
BD_ADDR	Wearable device Bluetooth address as shown in smartphone settings or master Bluetooth address.
bluetooth_version	Bluetooth version used.
android_app	Specific Android app used for data exchange, in this dataset used for smartwatches.
android_app_version	Version of android_app
master_device	Parameters regarding the master device, in this dataset Samsung Galaxy S20 FE.
recording_duration_seconds	Duration of recording in seconds.
recording_timeline_description	Events at which Android Debug Bridge triggered connection events in smartphone. Seconds.
enabling_Bluetooth_on_smartphone	Time in recording at which Bluetooth was enabled on smartphone. Seconds.
Bluetooth_connection_established	Time at which pairing with wearable device was complete. Seconds.
start_data_exchange	Time at which data exchange with Wearable device was triggered. Seconds.
stop_data_exchange	Time at which data exchange with Wearable device was stopped. Seconds.
disconnected	Time at which Android Debug Bridge triggered to turn off smartphone's Bluetooth. Seconds.
event_scenario	Event as described in Section 3.2. Paired, unpaired or advertising.
event_description	Description of what was happening during recording in free form.
file_format	File format of output_file.
channelized_data	Parameters regarding the channelized data.
channels_25	Parameters regarding the data channelized into four 25 MHz channels.
output_file_ch25	File names of channelized data with reference to metadata location. 25 MHz channels.
sample_rate_ch25	Sample rate of output_file_ch25.
center_frequency_ch25	Center frequencies of output_file_ch25 in same order as output_file_ch25.
channels_05	Parameters regarding the data channelized into twenty 5 MHz channels.
output_file_ch05	Filenames of channelized data with reference to metadata location. 5 MHz channels.
sample_rate_ch05	Sample rate of output_file_ch05.
center_frequency_ch05	Center frequencies of output_file_ch05 in same order as output_file_ch05.

Bluetooth is enabled on the smartphone at 1.2 to 1.5 s after start of recording, and until then, only the DUT transmits advertising packets. The length of different communication phases varies between unpaired and paired scenarios and between different devices because Bluetooth devices can take different amounts of time to establish a connection. The duration of Bluetooth connection establishment ranges from 8.5 to 20.0 s for unpaired scenarios and from 4.7 to 17.0 s for paired scenarios, but varies significantly from device to device. Consequently, the start and stop times of the data exchange and disconnection phases in the recording also vary. The actual data exchange between the Android device and the DUT is terminated after approximately 5 s. For devices other than headsets, the Bluetooth connection management is handled by device-specific Android apps, and in our

current setup, we cannot automatically determine when data exchange starts and stops for these devices.

### 3.4. Data Quality

To validate the recorded dataset for Bluetooth traffic analysis, we conducted Bluetooth data decoding similar to what would occur on the RF chip of a Bluetooth device. The analysis of the captured data involved the following steps:

1. Each 100 MHz wide recording was processed into four 25 MHz wide sample files with frequency shifts  $-30$ ,  $-10$ ,  $+10$ ,  $+30$  MHz, and decimated four times applying a Chebyshev type I filter of 10th order.
2. Each 25MHz wide sample file was processed into five 5 MHz wide sample files with frequency shifts  $-5$ ,  $-2.5$ ,  $0$ ,  $+2.5$ ,  $+5$  and decimated five times applying a Chebyshev type I filter of 10th order (each resulting sample file represents four Bluetooth Classic channels).
3. Radio signals were detected using amplitude peak detection. Detecting included anything that was longer than the smallest BTC packet length and had an amplitude above a hard-coded multiplier of the average noise level.
4. Based on the frequency with the highest amplitude, the specific Bluetooth Classic channel was extracted from the signal samples.
5. Performing Gaussian frequency shift keying (GFSK) demodulation with multiple possible symbol start time shifts, generating multiple candidate demodulation results.
6. Conducting correlation with the expected preamble, trailer, and the fixed bits of the access word (...001101 or 110010), and the result used to score the candidate demodulation alternatives.
7. Where applicable, the header forward error correction (FEC) was decoded, and its error rate was also used to adjust the candidate demodulation score.
8. The decoded bits of the highest-scoring demodulation candidate were recorded in a file of potential packets along with other metadata (sample start, length, channel, LAP, etc.).
9. Since the resulting bits are whitened, further processing was performed to decode packets, e.g., detection of FHS packets, page central response packets, validation of possible de-whitening by checking FEC calculation, etc. This allowed decoding of the packet type, upper address part (UAP), and further processing of decoded packets (e.g., recovering payload).

During our analysis, we discovered that signal detection could be conducted simultaneously across multiple nearby channels. With four channels, we observed no significant differences in results compared to processing individual channels, while also obtaining a performance gain. As there is a filter with roll-off on the sides, we chose a bandwidth of 5 MHz which included a 0.5 MHz “buffer zone” on the edges for filter roll-off. This allowed us to avoid the effects of too steep a filter roll-off. Theoretically, we could have downsampled directly from 100 MHz into 5 MHz wide files, but this processing was practically inconvenient due to constraints of processing power and host RAM, and the described two-step process for channel splitting enabled us to complete the processing faster.

The resulting list of decoded packets includes a varying level of metadata (described in Table 3) depending on the success of the decoding steps (e.g., successful detection of an FHS packet may affect the ability to correctly de-whiten the data). We did not implement further packet processing logic based on the packet type, such as phase shift keying (PSK) demodulation for packets with enhanced data rate (EDR) payload parts, but the length of amplitude-based detection of the signal should also include the full PSK payload in the selected packet length.

**Table 3.** Information about the decoded data.

Key	Description
Packet	Packet sequence number within the radio recording
sample_file	Path to the sample file containing the origin recording
left	Packet first sample in sample file for this packet
right	Packet final sample in sample file for this packet
length	right–left, length of the signal in samples
signal_max	Maximum amplitude of the detected signal
signal_mean	Signal mean amplitude
payload_std	Standard deviation of the signal amplitude. Calculated only if the overall length of the signal is sufficient for payload EDR payload
index_25	Which of the four 25 MHz channelized recordings
index_5	Which of the five 5 MHz channelized recordings
local_freq	Local frequency within the 5MHz sample file
demod_start	Offset of performed GFSK demodulation start from the signal start
bits	Decoded bits (if available)
packet_lt_addr	Bluetooth logical transport address (LT_ADDR)
packet_type	Header TYPE field
packet_flow	Header FLOW field
packet_arqn	Header ARQN field
packet_seqn	Header SEQN field
packet_hec	Header HEC field
packet_id	Is the packet an ID packet?
header_fec	Ratio of header FEC 1/3 bits, which are equal (higher number—more likely decoding is correct)
clock	Starting sample in sample_file for demodulated packet (left + demod_start)
lap	LAP (lower address part)
comment	Additional comments (if any)
header	Packet header

We were also able to validate that the recorded data can be decoded with the open source `ice9-bluetooth-sniffer` [7] (commit 2d504c7), but that required a modification of the squelch setting from the default  $-45$  dB to  $-22.2$  dB in its settings to improve the rate of packet detection. This resulted in over 1000 decoded packets for most of the recordings.

#### 4. User Notes

This section describes how to interact with the dataset, including two Python scripts for basic usage. Metadata, example radio and decoded data, and usage examples are available on GitHub <sup>1</sup> (accessed on 28 March 2024). It is not necessary to download the entire dataset to analyze a single recording, but we assume that the downloaded parts follow the folder structure described in Section 2. The following example Python scripts are provided to demonstrate the structure and use of the data.

`plot_data.py` is used to analyze the waveforms we provided in the dataset. It provides a basic user interface for selecting the recording of interest and generating time and frequency plots. The program will automatically set the correct sample rate and data type for both channelized options and non-channelized data. Please make sure you have enough free memory, as this program loads the entire selected radio recording into RAM. Figure 3 shows the output of `python3 plot_data.py Sony_WH-1000XM5/recording_1/paired/top.yaml`. This example demonstrates code to locate, select, and ingest the raw recording data.

`demodulated_data.py` demonstrates the demodulation and decoding results. It searches the entire dataset for the decoded data JSON files and prompts the user to select one of them. While providing valuable insight into wireless traffic, it is important to note that we used our own methods to decode the data and the results may not fully represent all traffic present. For instance, PSK demodulation, commonly used by Bluetooth devices, is not performed in our processing and, thus, is not included in the results. An example of the output of this script is shown in Figure 4. We consider that this demodulation and decoding validates the content of the recordings, demonstrating that the packet data can be recovered from the raw radio data samples.

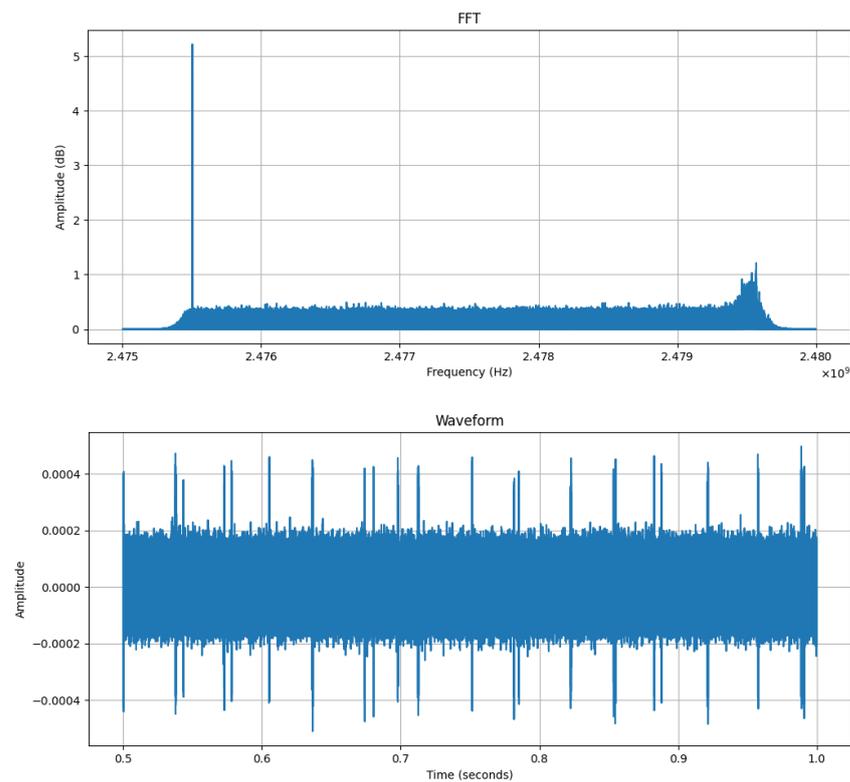


Figure 3. Example output of plot\_data.py program.

```

Select a JSON file to load:
1. ./_example_radio_data/_example_recording.json
Enter the number corresponding to the file you want to load: 1
Number of all packets: 4423
Enter the packet number you want to load (or 'q' to quit): 1000
Packet 1000:
sample_file: Beats_Solo3_Wireless/recording_1/paired/process/
radio_05_1_3.chdata
left: 12730371
right: 12730771
length: 400
signal_max: 0.0015389412
signal_mean: 0.0013515214
payload_std: -1.0
index_25: 1
index_5: 3
local_freq: -1512.5
demod_start: 38
bits: 01010100011101011100010110001100110001110011001101000101111001110010
packet_lt_addr: None
packet_type: ID
packet_flow: None
packet_arqn: None
packet_seqn: None
packet_hec: None
packet_id: True
header_fec: 0.0
clock: 12730409
lap: 110011001101000101111001
comment:
header: None

```

Figure 4. Example output of demodulated\_data.py program.

We consider that the dataset is suitable for the following types of research and experiments, enabling validation of algorithms on samples from a larger variety of relevant devices than might otherwise be available to researchers:

- Device model fingerprinting based on both physical layer and protocol aspects;
- Testing of algorithms for radio data analysis, packet detection and decoding;
- Exploration of Bluetooth protocol implementation differences in various chipsets;
- Vulnerability research on data encryption weaknesses based on observation of the pairing process.

**Author Contributions:** Conceptualization, K.N. and P.P.; methodology, A.R. and P.P.; software, A.R. and D.T.; validation, E.D. and A.R.; formal analysis, E.D.; investigation, A.R.; resources, K.N.; data curation, D.T. and A.R.; writing—original draft preparation, A.R. and E.D.; writing—review and editing, P.P. and E.B.; visualization, A.R.; supervision, P.P.; project administration, P.P. and K.N.; funding acquisition, P.P. and K.N. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research is funded by the Latvian Council of Science, project “Automated wireless security analysis of wearable devices” (WearSec), project No. lzp-2020/1-0395.

**Data Availability Statement:** The original data presented in the study are openly available in the Institute of Electronics and Computer Science data sharing platform at <https://pubfaili.edi.lv/wearsecdata> and metadata in Github at [https://github.com/edi-riga/Wearable\\_device\\_dataset/tree/main](https://github.com/edi-riga/Wearable_device_dataset/tree/main).

**Conflicts of Interest:** The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

## Abbreviations

The following abbreviations are used in this manuscript:

ADB	Android Debug Bridge
BD_ADDR	Broadcast Address
BLE	Bluetooth Low Energy
BTC	Bluetooth Classic
DUT	Device Under Test
EDR	Enhanced Data Rate
FEC	Forward Error Correction
GFSK	Gaussian Frequency Shift Keying
PSK	Phase Shift Keying
RF	Radio Frequency
SDR	Software Defined Radio
SoC	System-on-Chip
UAP	Upper Address Part

## Note

- <sup>1</sup> [https://github.com/edi-riga/Wearable\\_device\\_dataset/tree/main](https://github.com/edi-riga/Wearable_device_dataset/tree/main).

## References

1. Givehchian, H.; Bhaskar, N.; Herrera, E.R.; Soto, H.R.L.; Dameff, C.; Bharadia, D.; Schulman, A. Evaluating physical-layer ble location tracking attacks on mobile devices. In Proceedings of the 2022 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 22–26 May 2022; pp. 1690–1704.
2. Jagannath, A.; Jagannath, J. Embedding-Assisted Attentional Deep Learning for Real-World RF Fingerprinting of Bluetooth. *IEEE Trans. Cogn. Commun. Netw.* **2023**, *9*, 940–949. [[CrossRef](#)]
3. A. Jagannath, Z. Kane, J.J. RF Fingerprinting Needs Attention: Multi-task Approach for Real-World WiFi and Bluetooth. In Proceedings of the IEEE Global Communications Conference (GLOBECOM), Rio de Janeiro, Brazil, 4–8 December 2022.
4. Takanen, A.; DeMott, J.; Miller, C.; Kettunen, A. *Fuzzing for Software Security Testing and Quality Assurance Second Edition*; Artech House: Norwood, MA, USA, 2018.

5. Great Scott Gadgets. Ubertooth One. Available online: [https://ubertooth.readthedocs.io/en/latest/ubertooth\\_one.html](https://ubertooth.readthedocs.io/en/latest/ubertooth_one.html) (accessed on 8 January 2024).
6. Texas Instruments. 2.4 GHz Low-Power RF Transceiver. Available online: <https://www.ti.com/lit/ds/symlink/cc2400.pdf?ts=1704707041389> (accessed on 8 January 2024).
7. Mike Ryan. ice9-bluetooth-sniffer. 2022. Available online: <https://github.com/mikeryan/ice9-bluetooth-sniffer> (accessed on 16 January 2024).
8. Uzundurukan, E.; Dalveren, Y.; Kara, A. A database for the radio frequency fingerprinting of Bluetooth devices. *Data* **2020**, *5*, 55. [[CrossRef](#)]
9. Siddik, A.B.; Drake, D.; Wilkinson, T.; De Leon, P.L.; Sandoval, S.; Campos, M. WIDEFT: A corpus of radio frequency signals for wireless device fingerprint research. In Proceedings of the 2021 IEEE International Symposium on Technologies for Homeland Security (HST), Boston, MA, USA, 8–9 November 2021; pp. 1–7.
10. Jagannath, A.; Jagannath, J. RF-Fingerprint-BT-IoT: Real-world Frequency Hopping Bluetooth dataset from IoT devices for RF fingerprinting. *TechRxiv* **2022**, *9*, 940–949. [[CrossRef](#)]
11. Jagannath, A.; Kane, Z.; Jagannath, J. Real-world Commercial WiFi and Bluetooth Dataset for RF Fingerprinting. *IEEE Dataport* **2022**. [[CrossRef](#)]
12. Ettus Research. Ettus Research Products. Available online: <https://www.ettus.com/all-products/x310-kit/> (accessed on 8 January 2024).
13. Ettus Research. Ettus Research Products. Available online: <https://kb.ettus.com/CBX> (accessed on 8 January 2024).
14. Zenodo. Zenodo Frequently Asked Questions. Available online: <https://help.zenodo.org/faq/> (accessed on 9 January 2024).
15. Core Specification Working Group. Bluetooth Core Specification v5.4. 2023. Available online: <https://www.bluetooth.com/specifications/specs/core-specification-5-4/> (accessed on 8 January 2024).
16. Bluetooth SIG. Part B. Baseband Specification. Available online: <https://www.bluetooth.com/wp-content/uploads/Files/Specification/HTML/Core-54/out/en/br-edr-controller/baseband-specification.html> (accessed on 23 February 2024).
17. GNU Radio Project. GNURadio. Available online: <https://www.gnuradio.org/> (accessed on 15 January 2024).
18. Android Developers. Android Debug Bridge (adb). Available online: <https://developer.android.com/tools/adb> (accessed on 8 January 2024).

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.